USING `BIANCHIPERIODPOLS`

LEWIS COMBES

1. Introduction

Let $\Gamma = \mathrm{PSL}_2(\mathbb{Z}_K)$ for $K = \mathbb{Q}(\sqrt{-d})$, where $d \in \{1, 2, 3, 7, 11\}$. The `BianchiPeriodPols` code computes Bianchi modular forms by way of their period polynomials, by way of the cocycles in $Z^1(\Gamma, V_{k,l}^{a,b})$.

This code is used to compute examples in the paper [1]. **It has more functionality than is used in the paper. Not all outputs of this code are proven to be computationally correct. Your mileage may vary.**

It is also important to note that the computation of Hecke eigenvalue systems uses Gabor Wiese's package `ArtinAlgebras`.

For a given level $\Gamma_0(\mathfrak{n})$ and set of weights k, l, a, b, the code can compute the Bianchi cusp forms $S_{k+2,l+2}^{a,b}(\Gamma_0(\mathfrak{n}))$.

The repo consists of the following files:

- `ProjAction.m` — Computes the action of Γ on the projective line over $\mathbb{Z}_K/\mathfrak{n}$.
- `WeightAction.m` — Computes the action of Γ on the weight module $V_{k,l}^{a,b}$.
- `Space.m` — Computes the actual space $W_{k,l}^{a,b}$ of period polynomials.
- `Hecke.m` — Computes the Hecke action on $W_{k,l}^{a,b}$ and collects eigenvalues with period polynomials.
- `PeriodPols.m` — Contains a few axillary functions to extract period polynomials.
- `H2.m` — Computes the space $H^2(\Gamma, V_{k,l}^{a,b})$ and the Hecke action there. This is used to find forms in H^2 for pairing with forms from H^1. This code is essentially the same as unpublished code of Haluk Şengün; see [3].

- `check_congruences.m` — Contains the Hecke eigenvalues of the four forms appearing in Section 5 of [1] for primes of norm up to 1000. Also contains code to check these are congruent modulo the claimed ideals.
- `loading_script_pols.m` — Contains general-use code for computing the period polynomials and Hecke eigenvalues attached to forms in a space defined by a level, weight, etc. **This is the most end-user-friendly place to start with this code.**

1.1. **Acknowledgement.** The code to compute Heilbronn matrices was written and shared with me by Haluk Şengün. The functions `detect_hnf` and `HNF_basis` were also written by him. His (unpublished) programs also helped me writing specific functions, including `GET_EV`, `WeightMat` and `ProjMat`. More generally, I could not code in Magma when I started my Ph.D. under Haluk's supervision, and it would have been a much harder journey from there to here without his guidance and support. Even though I wrote the bulk of the code here, he is nonetheless present in every line, and I am grateful for his endless reservoirs of support.

2. Getting started

The main utility of the code is computing the polynomials and Hecke eigenvalues for a specific level and weight. This can be done with `loading_script_pols.m`. The parameters one needs to set are the following:

- `d` — The d in $K = \mathbb{Q}(\sqrt{-d})$. For $d \notin \{1, 2, 3, 7, 11\}$, the code will return an error, as these are not implemented.
- `level` — This is a list of length 2, corresponding to a generator of an ideal $\mathfrak{n} \triangleleft \mathbb{Z}_K$. For the above d, all the $\mathbb{Z}_K$ are principal ideal domains, so only one generator needs to be specified. The bases used by this parameter are
  - $d = 1$: $\mathbb{Z}_K = \left\{1, \sqrt{-1}\right\}$
  - $d = 2$: $\mathbb{Z}_K = \left\{1, \sqrt{-2}\right\}$
  - $d = 3$: $\mathbb{Z}_K = \left\{1, \frac{1+\sqrt{-3}}{2}\right\}$
  - $d = 7$: $\mathbb{Z}_K = \left\{1, \frac{1+\sqrt{-7}}{2}\right\}$
  - $d = 11$: $\mathbb{Z}_K = \left\{1, \frac{1+\sqrt{-11}}{2}\right\}$

For, for example, if $d = 2$, the parameter `level:=[3,5]` corresponds to the ideal $(3 + 5\sqrt{-2})$.

- `weight` — A list of length 4, corresponding to a given weight module. The entries of the list `[k,l,a,b]` correspond to the weight module $V_{k,l}^{a,b}$. The k and l give the dimensions of its component parts, and the a and b are the determinant twists (see [4], Section 5). To compute spaces without twists, set $a = b = 0$. For modular forms (conjecturally) attached to elliptic curves over K, one needs to use the trivial weight `[0,0,0,0]`.
- `char` — The characteristic of the cohomology groups computed. One interesting aspect of Bianchi modular forms is that mod p classes in the cohomology of Bianchi groups don't always lift to characteristic 0. This parameter should be set to `0` to compute with actual modular forms. To compute with mod p modular forms, set this to `p`.
- `HB` — A number that bounds the Hecke operators computed. If `HB:=50`, then all the Hecke operators for primes `p` with $N(\mathfrak{p}) \leq 50$ are computed.
- `chi` — A Dirichlet character associated to the level `n`, given by an integer. The integer `r` specifies the character as entry `r` in the list `Elements(DirichletGroup(level))`.
- `type` — A string that specifies whether we want to cut down to the + space of the "Hecke operator at infinity"[1], or just take the whole space. These correspond to the strings `GL` and `SL` respectively. Any other string will return an error. In general, people seem to mostly only care about `GL` forms.
- `optimizeHF` — A boolean value that lets one specify whether to optimise the fields in which the Hecke eigenvalues are returned. When the characteristic is 0, this is done with Magma's function `OptimizedRepresentation`, which computes maximal orders in number fields, which can be costly. In general, this should not be a problem for small levels and weights. When the characteristic is p, optimisation is done by running over subfields of $\mathbb{F}_{p^n}$, which is relatively cheap.

Once these have been set, `loading_script_pols.m` does the rest.

[1] given by the action of the matrix $J = \left(\begin{smallmatrix} \varepsilon & 0 \\ 0 & 1 \end{smallmatrix}\right)$, where ε generates the unit group of $\mathbb{Z}_K$

3. Examples

3.1. **An eigenvalue system attached to an elliptic curve.** First, let's check the code reproduces known eigenvalue systems. We look for the modular form `2.0.8.1-32.1-a`. Per the LMFDB, this is a weight 2 modular form of level $\mathfrak{n} = (4\sqrt{-2})$ over $\mathbb{Q}(\sqrt{-2})$. This translates to the following parameters for our code:

- `d:=2;`
- `level:=[0,4];`
- `weight:=[0,0,0,0];`
- `char:=0;`
- `HB:=30;`
- `chi:=1;`
- `type:="GL";`

All the results of the computation are bundled up into the Magma record `W`. We can check the dimension of the returned space with[2] `W‘dim` or `Dimension(W‘space)`. In this case, the dimension is 8.

Next, running the `if` statement underneath computes the Hecke eigenvalues. In this case, the space splits up into three eigenvalue systems.

$\mathfrak{p}$	$[1,1]$	$[-1,1]$	$[3,1]$	$[3,-1]$	$[3,-2]$	$[3,2]$	$[-1,3]$	$[-1,-3]$	$[5,0]$
$N(\mathfrak{p})$	3	3	11	11	17	17	19	19	25
$e_1(\mathfrak{p})$	4	4	12	12	18	18	20	20	26
$e_2(\mathfrak{p})$	-4	-4	-12	-12	18	18	-20	-20	26
$a(\mathfrak{p})$	0	0	0	0	2	2	0	0	-6

Figure 1. Eigenvalue systems over $\mathbb{Q}(\sqrt{-2})$ at level $(4\sqrt{-2})$, weight 2.

[2] LaTeX's `\texttt` environment renders this slightly strangely, but for clarity: the character between `W` and `dim` is the backtick. It is found on UK QWERTY keyboards underneath the escape key, and is unicode `U+0060`.

These are returned in the list `EV_systems`. Each entry is a tuple with three parts. These are, in order, a polynomial defining a number field in which the period polynomial lives, the multiplicity of the system, and the eigenvalues themselves. The eigenvalues are in the order that the primes appear in the list `HP`.

In this case, the code says that e_1 appears with multiplicity 5, e_2 appears with multiplicity 2, and a appears with multiplicity 1. Checking against the LMFDB, the eigenvalue system we are looking for is clearly the multiplicity 1 system a. The systems e_1 and e_2 are Eisenstein.

We can also see the period polynomial attached to this system. In the case of a form with non-trivial level, the period polynomial is really the *extended period polynomial* (see [2], Remark 2.2), which assigns a period polynomial to each of the elements of $\mathbb{P}^1(\mathbb{Z}_K/\mathfrak{n})$. In this case, since the weight module is trivial, each of these period polynomials is really just a number.

The period polynomials are returned by the function `GetPolVals`, which represents them each as a list with three entries. The entries are, in order, a vector representing the period polynomial, a list containing the Hecke eigenvalues, and a list containing the generators of the prime indices of the eigenvalues.

The (extended) period polynomial attached to the system a is `(1 -1 1 -1 0 0 0 0 2 2 0 0 -2 0 -2 0 -1 0 0 0 0 0 0 1 2 0 -2 -2 -2 0 2 2 2 0 -2 -2 -2 0 2 2 0 0 0 -1 0 0 0 1)`, which is extremely unedifying to simply look at. By the Eichler-Shimura isomorphism, each of these numbers is related to an integral of the underlying modular form F that gives rise to the eigenvalue system a. Of particular interest is the period polynomial attached to the identity component of the projective line $\mathbb{P}^1(\mathbb{Z}_K/\mathfrak{n})$. This is the point $(0:1)$, and its position in the projective line be found using `W‘id_index`. The period polynomial here is comprised of L-values attached to F, and in this case there is only 1 value of interest, which is $L(F,1)$. The first entry in the vector is `1`, so we know $L(F,1) \neq 0$ for this form[3]. This matches the LMFDB, which has the analytic rank of this form to be 0.

[3]This is about all we can tell from this computation. The "correct" scaling of the L-value in cohomology is a tricky issue we choose to (mostly) ignore.

When one has level and weight, it is harder to just read off the period polynomial from the vector. In these cases, the function `PeriodPol` can be used. Inputting the command `PeriodPol(W,pol_vals[6][1])` returns the 1×1-matrix `[1]`. We return period polynomials as matrices because it is easier to see some of their properties that way. For further details, see Remark 5.2 in [1].

One can also use the optional argument `ind` in the function `PeriodPol` to get polynomials attached to different elements of $\mathbb{P}^1(\mathbb{Z}_K/\mathfrak{n})$. For example, if we want the period polynomial corresponding to the 13th entry of the projective line, the command `PeriodPol(W,pol_vals[6][1] : ind:=13)` gives `[-2]`. This functionality becomes more relevant when one combines non-trivial levels and weights.

3.2. **Eigenvalue systems from** [1]**.** To compute the examples in the paper, we use the parameters

- `d:=11;`
- `level:=[1,0];`
- `weight:=[10,10,0,0];`
- `char:=0;`
- `HB:=30;`
- `chi:=1;`
- `type:="GL";`

In this case, there is no projective line, and the whole vector returned in `pol_vals` for each eigenvalue system is the period polynomial.

To compute the pairing in [1], one can use the code in `example.m`. For two polynomials `p1` and `p2`, the pairing can be computed with `PairPol(weight,p1,p2)`. The input of `weight` is necessary, since the polynomials themselves have no intrinsic notion of the space of homogeneous polynomials they live in for our purposes.

The pairing is only defined in [1] for forms of level 1. The analagous pairing for forms with level $\Gamma_0(\mathfrak{n})$ is likely to take the form

$$\frac{1}{[\Gamma : \Gamma_0(\mathfrak{n})]} \sum_{\alpha \in \mathbb{P}^1(\mathcal{O}/\mathfrak{n})} \langle r_F^\alpha, v_G^\alpha \rangle$$

where r_F^α is the component of the extended period polynomial coming from the projective line element α, see [2, Section 3]. But we have not proved this, or implemented it.

4. Misc. notes

4.1. **`W‘spec`.** At a certain point in coding, it became convenient to store all the invariants defining the space in one place. This is done in `W‘spec`. It is passed to certain functions during the computation of the space of period polynomials, and is likely worth ignoring entirely if one is using this code as an "end-user".

4.2. **Projective line.** Magma's projective line function very occasionally returns different orderings of its elements. A previous version of the code simply recalled `ProjectiveLine` every time it was needed, but this marginal discrepancy was causing compatibility problems between matrices. So now we simply conjure the projective line once, in `Space.m`, and reuse it whenever we need it. It is stored in `W‘spec‘PL`, and its recognition function is stored in `W‘spec‘r`.

4.3. **Characters.** I am not currently completely convinced characters are working correctly. This was implemented as part of experiments with Serre's conjecture for mod p representations over imaginary quadratic fields, and in the course of those experiments I came across anomalies I can't yet explain. Such anomalies are almost always due to errors in the code. It's a feature I haven't spent much time testing, so **your mileage may vary**. Taking the functionality out of the code is more trouble than it's worth, as I plan to return to it once I have some time.

References

[1] L. Combes. *Bianchi period polynomials: Hecke action and congruences.*

[2] V. Paşol & A Popa. *Modular forms and period polynomials.* Proc. London Math. Soc. (3) 107 (2013), 713–743.

[3] M.H. Şengün. *On the integral cohomology of Bianchi groups.* Exp. Math. 20 (2011), no. 4, 487–505.

[4] R. Torrey, *On Serre's conjecture over imaginary quadratic fields.* Journal of Number Theory, Volume 132, Issue 4, 2012, Pages 637-656.